

Remarks

This Amendment responds to the final Office Action (“the Action”) mailed August 16, 2006. Reconsideration of the application is respectfully requested in view of the foregoing amendments and following remarks. Claims 1-36 are pending in the application. No claims have been canceled. No claims have been allowed. Claims 1, 21, 25, 27, 29, and 32 are independent.

Cited Art

U.S. Patent Application No. 11/330,053 (Pub. No. 2006/0129994) to Srivastava et al. (“the ’053 application”) is entitled “Method and Apparatus for Prioritizing Software Tests.”

The February 2002 publication by Srivastava et al. is entitled “Effectively Prioritizing Tests in Development Environment” (“Srivastava”).

Amendments

Claims 1, 12-15, 21, 25, 27, and 29 have been amended to clarify claim language. No new matter has been added.

Claim Objections

The Action objects to apparent inconsistent punctuation in claim 29. Claim 29 is now amended to recite a colon after the recited “software stored in memory comprising computer executable instructions for” text. Additional amendments have also been made to claim 29 in order to provide consistent punctuation. Applicants request that the objection to claim 29 be withdrawn.

The Action also objects to the use of the language “system” and “subsystem” in the claims, alleging that the scope of the terms is unclear. Applicants respectfully address the scope of this language in the discussion of the Action’s rejection of the claims under 35 U.S.C. § 112, which can be found below.

Rejections under 35 U.S.C. § 101

The Action rejects claims 1-11 and 16-31 for being directed to non-statutory subject matter. Although the Applicants respectfully disagree with the rejection, claims have been amended to highlight that the claims are directed to statutory subject matter.

With respect to the rejection of claims 1-11 and 16-20, claim 1 has been amended to recite “displaying a representation of the collection.” This language was previously recited in claims 12-15, which the Action found to be directed toward statutory subject matter. Claims 12-15 are also amended as appropriate to properly depend from the amended claim 1. Applicants believe that claim 1 as amended, and as well as dependent claims 2-20, are thus directed toward statutory subject matter.

With respect to claims 21 and 25, the claims have been amended to recite actions of performing tests to produce test results. For example, claim 25 has been amended to recite “means for performing prioritized tests according to test priorities to produce test results.” Applicants believe that claims 21 and 25 as amended, and thus claims dependent claims 22-24 and 26-28 as well, are thus directed toward statutory subject matter.

Finally, with respect to claims 29-31, applicants note that claim 29, as filed, recites “a computer system” comprising a processor and memory, as well as software stored in memory. Thus, claim 29 is not directed toward an abstract method or software *per se*, but rather to a computer apparatus. Applicants believe that claim 29, and thus dependent claims 30 and 31, are thus directed toward statutory subject matter.

Applicants respectfully request that the rejections of claims 1-11 and 16-20 under 35 U.S.C. § 101 be withdrawn and the claims allowed.

Rejections under 35 U.S.C. § 112

The Action rejects claims 1-36 as being indefinite. In particular, the Action alleges that “claims 1-36 exhibit the use of ‘system’ and ‘subsystem’ without specifying what system or subsystem this is all about.” [Action, at page 6, § 10.] The Action also cites specific language in claims 1, 21, 27, and 29 to reject under 35 U.S.C. § 112. [*Id.*]

With respect to the terms “system” and “subsystem,” the Applicants respectfully disagree with the rejection in the Action and note that these terms are given specific and definite

meanings in the specification. For example, the Application describes examples of systems and subsystems:

Figure 3 shows exemplary abstractions system division 300. In this exemplary abstraction, a system 300 is a collection of subsystems 302-308, and a subsystem is a collection of binary files 310-314.

[Application, at page 7, lines 19-21; Figure 3.] A more detailed example is given on page 5:

Figure 1 shows an overview of a system 100 with dependent subsystems. In the modern computing environment, several subsystems 102-108 are interdependent. Any individual subsystem such as graphical and operating services 104 may individually be very large, but is typically also dependent on the services provided by other subsystems. For example, a subsystem 104 provides graphical and operating services (e.g., Microsoft ® Windows™), that are utilized by other subsystems 102, 106, 108. Similarly, a database subsystem 106 (e.g., Microsoft ® SQL Server™), provides services that other subsystems may need from time to time. Services are provided, for example, via one or more binary files (e.g., .dll, .exe, etc.). A subsystem is a logical collection of one or more binary files ("binaries"). For example, the Microsoft ® Windows™ operating system subsystem contains hundreds of binary files such as kernel.dll, gdi.dll, and user.dll. Together the subsystems provide the aggregate services needed for the computing system 100.

[Application, at page 5, lines 3-15; Figure 1.] Thus, as the Application states, subsystems are collections of binary files and systems are collections of subsystems.

Applicants note that the Action alleges that the system "can be a system of abstractions as collected in the interface; or a system of elements in test/binary files." [Action, at page 6, § 10.] However, the Application, continuing from the quote of page 7 above, takes care to explain a use of logical abstractions:

Logical abstractions exist for many reasons, and often help reduce complexity for human understanding. For example, binary files may be grouped into subsystems because they have some common overall function they support. In one example, a subsystem supports word processing, and programmers writing the word processing software are assigned to the team writing word processing software. In such a case, it can be helpful to view the binary files in the subsystem as "word processing" software, so a word processing team can be managed as a group. Such an abstraction may also be functional in nature, since the word processing files may be released according to customer word processing needs.

However, other levels or views of abstractions would just as easily be implemented by the described technologies. For example, the subsystem abstraction may not be required, if all binary files are viewed as part of the system.

[Application, at page 7, line 25 to page 8, line 10.] Thus, while abstractions are useful in terms of understanding dependencies and groups of files, this does not change the examples given above for the meaning of “system” and “subsystem.”

With regard to the rejection of claims 1 and 27 under 35 U.S.C. § 112, the Action rejects the claims for insufficient antecedent basis for the term “subsystem.” Applicants note that claims 1 and 27 have been amended to address this rejection. For example, claim 1 now recites “logical abstractions outside of a subsystem for the target logical abstraction,” which does not rely on language without proper antecedent basis. For at least these reasons, claims 1 and 27, as well as dependent claims 2-20 and 28, are allowable. Applicants request that the rejection of claims 1-20, 27, and 28 under 35 U.S.C. § 112 be withdrawn and the claims be allowed.

With regard to the rejection of claims 21 and 29 under 35 U.S.C. § 112, the Action rejects the claims for insufficient antecedent basis for the term “subsystem dependency information” with no mention of a particular system. While Applicants disagree that a particular system must be recited in order to provide antecedent basis for the quoted language, in the spirit of expediting examination Applicants have amended claims 21 and 29 to address this rejection. For example, the claims now each recite “subsystem dependency information *for a subsystem of a system.*” For at least these reasons, claims 21 and 29, as well as dependent claims 22-24 and 30-36, are allowable. Applicants request that the rejection of claims 21-24 and 29-36 under 35 U.S.C. § 112 be withdrawn and the claims be allowed.

Rejections Under 35 U.S.C. § 102 over Srivastava

The Action rejects claims 1-36 under 35 U.S.C. § 102(b) as being anticipated by Srivastava. Applicants respectfully submit the claims in their present form are allowable over the cited art. For a 102(b) rejection to be proper, the cited art must show each and every element as set forth in a claim. (*See* MPEP § 2131.01.) However, the cited art does not describe each and every element. Accordingly, applicants request that all rejections be withdrawn. Claims 1, 21, 25, 27, 29, and 32 are independent.

Claim 1

Claim 1, as amended, recites:

receiving a system definition and a request for dependency information;

requesting, via an application programming interface, a system dependency creation request comprising the received system definition and a dependency request comprising a target logical abstraction;

receiving, responsive to the application programming interface request, a dependency collection for the target logical abstraction comprising logical abstractions in one or more dependency chains with the target logical abstraction, wherein the dependency collection comprises logical abstractions outside of a subsystem for the target logical abstraction;

[emphasis added.] The application, provides examples system definition files:

Figure 6 shows an example system definition file. In this case, the system definition file is represented as an XML file 600. The abstraction levels in this example are defined as system 602, subsystem 606, and binary (file) 608. In this example, *the system definition file identifies the universe of desired dependencies by indicating the names 608 of the input binary files, and the name 608 of the XML file where the binary file dependency relationships are stored.* Also, the example shows a subsystem name 606, and the name 610 of the XML file where the subsystem dependency relationships are stored. . . . *The dependency information is stored in XML files (e.g., 610, 614) according to the levels of abstraction of an example system. Other examples could group dependency information in different arrangements so long as the information is stored for dependency mining.*

[Application, at page 13, line 25 to page 14, line 9.] Further, the Application gives examples of the use of system definition files:

A dependency framework 202 receives a system definition (not shown) which defines one or more subsystems 204, 206, 208, 210. The system definition describes the subsystems and the binary files within each subsystem. The system definition input can be created, for example, via a graphical user interface. It can also be received by the framework as an input file. The dependency framework uses the system definition to determine a universe from which to discover binary dependencies. The dependency framework discovers what binaries depend on other binaries in providing services.

[Application at page 6, lines 14-21.]

Srivastava performs test prioritization by directly analyzing binary files, and thus does not, and has no need to, describe a system description as recited in claim 1. In its rejection of claim 1, the Action alleges that the “system definition” of claim 1 is described at, among other locations, Figure 1, and pages 2 and 3 of Srivastava. However, as the cited passages make clear, Srivastava’s “Echelon” system uses executable binary files, rather than system definition files, to perform its analysis. For example, Figure 1 illustrates “Old Binary” and “New Binary” as input

into the “Binary Change Analysis” module. Srivastava makes this clear in its description accompanying Figure 1:

Echelon takes as input two versions of a program in binary form along with the test coverage information of the old binary and produces a prioritized list of the tests, as well as a list of the modified and new blocks (or source files) that may not be executed by any existing test. Echelon accomplishes this task in three steps illustrated in Figure 1.

In the first step, Echelon uses BMAT to find a matching block the old binary for each block in the new binary.

[Srivastava, at page 3, left col., para. 5 to page 3, right col., top para.]

Thus, the system of Srivastava is directed toward direct analysis of binary executable files rather than usage of a “system definition” as is recited in claim 1. Furthermore, Srivastava in fact teaches the usefulness of performing analysis on binary files. See, for example, Srivastava’s description of the benefits of the BMAT tool at page 3, left column, fourth paragraph, as well as Srivastava’s description of one of the “key features” of Echelon:

Echelon operates at the binary level making it easier to integrate into the development process. Echelon scales to real product binaries in large-scale development environments.

[Srivastava, at page 2, left col., para. 6.] Thus, by encouraging the use of binary files for test prioritization analysis, Srivastava not only does not describe the recited language of claim 1, but actually *teaches away* from usage of a system definition, which provides a degree of abstraction that direct binary analysis would not.

Srivastava’s Echelon system, by operating on programs, is not flexible enough to teach or suggest system-level information, such as, for example, the language “a system dependency creation request comprising the received system definition” as recited in claim 1. The Application describes an example of the usefulness of considering systems and subsystems over a single-minded focus on individual programs:

Programs are rarely self-contained in real software environments. They depend on other programs or shared subsystems like language run time and operating system libraries for various functionalities. These subsystems are developed external to the program, with their own test and development process. However, a change in one of the external subsystems may affect the program and one or more other external subsystems.

As a result, many users are reluctant to upgrade to newer versions of various software components as they fear that some dependent subsystems may

stop working. Further, software development teams don't have the information they need to make informed decisions not only about the risks posed by changes made to subsystems they depend on, but risks they pose to other subsystems by changing their own subsystem.

Historically, techniques have been proposed for test selection and test prioritization to reduce the cost of testing. However, these proposed techniques focus internally only on a program. For example, they consider internal parameters such as changes made to the program itself, rates of faults in the program, and block coverage of the program.

[Application, at page 1, line 15 to page 2, line 3.] And thus, claim 1 recites language such as, for example, "receiving a system definition" and "requesting . . . a system dependency creation request comprising the received system definition . . .," rather than focusing solely on program analysis.

However, as noted in the above-quoted language from page 3, Srivastava's Echelon system uses for its input two versions of *a program* in binary form. Srivastava's concentration on program analysis is further highlighted by the Action's citation used in its rejection of the "system dependency creation request comprising the received system definition" of claim 1. In its rejection, the Action cites to a description of the BMAT tool. However, as the cited language shows, the BMAT tool is clearly focused on analysis of programs, rather than systems or subsystems:

To compute the changes between two versions of the program, Echelon utilizes BMAT, a binary matching tool BMAT is a fast and effective tool that matches two versions of a binary program without the knowledge of source code changes.

[Srivastava, at page 3, left col., para. 4.] Thus, Srivastava is directed toward analysis of individual binary programs, and thus cannot teach or suggest the language of claim 1 recited above.

For at least these reasons, Srivastava does not teach or suggest the above-recited language of claim 1 and thus Srivastava does not describe each and every element of claim 1. Claim 1, as well as claims 2-20, which depend from claim 1, are thus allowable and applicants request their allowance.

Claim 21

Claim 21 recites, in part:

propagating dependency information to determine subsystem dependency information for a subsystem of a system;
propagating the subsystem dependency information to determine system dependency information for the system; . . .

In its rejection of claim 21, the Action, in reference to the above-quoted language, cites to a passage of Srivastava which it cited in its rejection of claim 1. Thus, for at least the reasons discussed above with respect to claim 1, Srivastava does not teach or suggest at least the above-recited language of claim 21 and thus Srivastava does not describe each and every element of claim 21. Claim 21, as well as claims 22-24, which depend from claim 21, are thus allowable and applicants request their allowance.

Claim 25

Claim 25 recites, in part:

means for determining binary dependencies for a defined system;
means for propagating binary dependencies to identify binaries dependent on binaries in other subsystems; . . .

In its rejection of claim 25, the Action, in reference to the above-quoted language, cites to a passage of Srivastava which it cited in its rejections of claims 1 and 21. Thus, for at least the reasons discussed above with respect to claim 1, Srivastava does not teach or suggest at least the above-recited language of claim 25 and thus Srivastava does not describe each and every element of claim 25. Claim 25, as well as claim 26, which depends from claim 25, are thus allowable and applicants request their allowance.

Claim 27

Claim 27 recites, in part:

creating a system definition in response to receiving graphical user interface input;
. . .
requesting via an application programming interface exposed by a dependency framework, a system dependency creation request comprising the system definition, and a target logical abstraction identifiable from the dependency information request; . . .

In its rejection of claim 27, the Action, in reference to the above-quoted language, cites to passages of Srivastava which it cited in its rejection of claim 1. Thus, for at least the reasons discussed above with respect to claim 1, Srivastava does not teach or suggest at least the above-recited language of claim 27 and thus Srivastava does not describe each and every element of claim 27. Claim 27, as well as claim 28, which depends from claim 27, are thus allowable and applicants request their allowance.

Claim 29

Claim 27 recites, in part:

propagating dependency information to determine subsystem dependency information for a subsystem of a system; and
propagating subsystem dependency information to determine system dependency information for the system;

In its rejection of claim 27, the Action, in reference to the above-quoted language, cites to its rejection of claim 21. Thus, for at least the reasons discussed above with respect to claims 21 and 1, Srivastava does not teach or suggest at least the above-recited language of claim 29 and thus Srivastava does not describe each and every element of claim 29. Claim 29, as well as claims 30 and 31, which depends from claim 29, are thus allowable and applicants request their allowance.

Claim 32

Claim 32 recites, in part:

means for accepting a system definition comprising binary files in plural subsystems

In its rejection of claim 32, the Action, in reference to the above-quoted language, cites to passages of Srivastava which it cited in its rejection of claim 1. Thus, for at least the reasons discussed above with respect to claim 1, Srivastava does not teach or suggest at least the above-recited language of claim 32 and thus Srivastava does not describe each and every element of claim 32. Claim 32, as well as claims 33-36, which depend from claim 32, are thus allowable and applicants request their allowance.

Double Patenting Rejection

The Action provisionally rejects claims 21, 25, and 29 under the judicially created doctrine of obviousness-type double patenting as being unpatentable over the '053 application in view of Srivastava. Applicants respectfully submit the claims in their present form are patently distinct over the cited art. To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. (MPEP § 2142.)

In the course of the provisional double patenting rejection, the Action admits that the '053 application does not recite a number of features, including:

determining dependency information about binary files, propagating such information to determine subsystem and system dependency, marking changed and unchanged logical abstraction to prioritize tests.

[Action, at page 3, paragraph 3.] Instead, the Action finds these features in Srivastava, relying on the same and similar passages as used in the rejections of claims 21, 25, and 29 under 35 U.S.C. §102, which were discussed above.

Applicants respectfully note that, for at least the reasons given above with respect to claims 21, 25, and 29, Srivastava does not teach or suggest at least one of the features of each claim. For example, Srivastava does not teach or suggest “propagating dependency information to determine subsystem dependency information for a subsystem of a system” and “propagating the subsystem dependency information to determine system dependency information for the system” as recited in claim 21. Because language from each of claims 21, 25, and 29 is found neither in the '053 application nor in Srivastava, such language is not taught or suggested by the combination of the two.

For at least this reason, the Action fails to make a establish a *prima facie* case of obviousness over the '053 Application in view of Srivastava. Accordingly, applicants request that the provisional double patenting rejection be withdrawn.

Request for Interview

If any issues remain, the Examiner is formally requested to contact the undersigned attorney prior to issuance of the next Office Action in order to arrange a telephonic interview. It is believed that a brief discussion of the merits of the present application may expedite prosecution. Applicants submit the foregoing formal Amendment so that the Examiner may fully evaluate Applicants' position, thereby enabling the interview to be more focused.

This request is being submitted under MPEP § 713.01, which indicates that an interview may be arranged in advance by a written request.

Conclusion

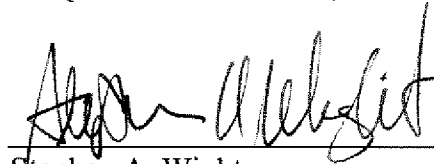
The claims in their present form should now be allowable. Such action is respectfully requested.

Respectfully submitted,

KLARQUIST SPARKMAN, LLP

One World Trade Center, Suite 1600
121 S.W. Salmon Street
Portland, Oregon 97204
Telephone: (503) 595-5300
Facsimile: (503) 595-5301

By



Stephen A. Wight
Registration No. 37,759